BlueMaster: Bypassing and Fixing Bluetooth-based Proximity Authentication

Youngman Jung
Samsung Electronics

Junbum Shin
Samsung Electronics

Yeongjin Jang
Oregon State University

Abstract

Bluetooth enabled devices can indirectly check the proximity of other connected devices, and this proximity check can be used as an authentication means. Thanks to the widespread use of Bluetooth, popular software vendors such as Google and Microsoft offer this device proximity authentication method in their operating systems, namely, Android and Windows 10. On one hand, Google's Android supports a feature called Android Smart Lock, which allows a user to register *trusted* Bluetooth devices, and then utilize the presence of such trusted devices as an alternative to passcode. On the other hand, Microsoft Windows uses this proof-of-device-proximity in a reverse way. Windows 10 introduces Dynamic Lock, which automatically *locks* the device if any of the paired Smartphone moves away, to block access to the computer while the user is unattended.

In this talk, we present the security pitfalls of Bluetooth-based proximity authentication. We analyze implementations of Android Smart Lock and Windows Dynamic Lock, and demonstrates new attacks on these implementations. Based on our analysis, we discovered three new attacks that allow attackers to bypass device proximity authentication. From Android Smart Lock, we discovered an attack to bypass a security check that prevents a basic MAC spoofing attack. From Windows Dynamic Lock, we discovered an attack to spoof a paired smartphone by altering the MAC address and device class. Moreover, these authentication methods are also vulnerable to a proximity spoofing attack.

Our analysis result shows that these vulnerabilities are originated from accepting untrusted data from Bluetooth for authentication. More severely, regarding the proximity checking, none of both is secure; Android ignores device proximity, and Windows is susceptible to signal amplification attack.

Finally, we discuss potential countermeasures and inherent weaknesses of proximity checking in Bluetooth as well as a method to analyze the security of the Bluetooth-based device and proximity authentication. Our countermeasure concludes with accepting only the trusted data from Bluetooth for authentication methods. Further, to help developers eliminate potential vulnerabilities in their development stage, we construct a method for analyzing the (in)security of the device proximity authentication methods and release a vulnerability detection tool for the problems we found.

1 Introduction

Bluetooth enabled devices can indirectly check the proximity of other connected devices, and this proximity check can be used as an authentication means. In particular, a host device can check the proximity of a user-registered Bluetooth device as an alternative to the user authentication. In other words, if a device is close enough to the host, the host can consider that device as the user's physical presence and grant the access to the host. Thanks to the widespread use of Bluetooth, popular software vendors such as Google and Microsoft offer this device proximity authentication method in their operating systems, namely, Android and Windows 10.

Smartphones running Android have enabled device proximity authentication via Android Smart Lock [1]. This mechanism allows the smartphone to check the proximity of a pre-registered trusted device and unlock the phone without entering the passcode if the device is close to the phone. Similarly, desktop and laptop computers running Windows 10 have enabled device proximity authentication via Windows Dynamic Lock [2]. However, this mechanism works in reverse. In other words, Windows Dynamic Lock aims to detect the user absence to block access to the computer while user is unattended. In doing so, this mechanism attempts to detect any displacement of the user's trusted device, and if detected, Windows considers that as a user absence and locks the device.

Despite the popular reliance on Bluetooth in device proximity authentication, surprisingly, the Bluetooth specifications [3–6] do not provide any security guarantee for the proximity authentication [1]. In particular, the Bluetooth

[1] The specifications describe about measuring the proximity of one device from the other, however, they do not provide any security feature for authenticating the device proximity.

specification defines security features for establishing secure communication between two endpoints, and software vendors implement the Bluetooth-based device proximity authentication on top of such secure communication foundations. For this reason, software vendors are required to implement an additional logic that handles the device and proximity authentication securely, and having a mistake in this logic could result in vulnerability in the authentication.

For instance, Martin Herfurt [7] discovered a MAC spoofing attack against Android Smart Lock in 2015. Additionally, Beccaro and Collula [8] also demonstrated how Android Smart Lock makes a mistake on authenticating a device. Not surprisingly, the root cause of the vulnerability was in the logic of implementing device authentication on top of the Bluetooth protocol. Specifically, Android Smart Lock authenticates a device via Service Discovery Protocol (SDP), which is an insecure channel of Bluetooth, and thereby, the authentication was insecure.

In this paper, we aim to analyze the security of Bluetooth-based device proximity authentication methods. Based on the observations of previously disclosed vulnerabilities and the fact that these authentications are not backed by the security features of Bluetooth, we take an action to address the core of the problem to help developers to eliminate potential vulnerabilities when developing device proximity authentication methods.

To this end, we construct a method for analyzing the (in)security of the Bluetooth-based device and proximity authentication method. First, we define trusted/untrusted properties of a Bluetooth connection by following the guidelines for Bluetooth security by NIST [6]. The primary rule in using such properties is that the authentication decision must rely on trusted properties. Second, to understand the authentication workflow, we model a state diagram of the authentication by applying reverse-engineering to the implementation. In particular, we focus on modeling two state diagrams, one is the state transitions for the Bluetooth connection between devices, and the other is for the logic that processes the device and proximity authentication. Finally, we discover potential vulnerabilities by analyzing these state diagrams. Specifically, we regard any state transition to a successful authentication state initiated by any untrusted property in a Bluetooth connection as a potential vulnerability.

After the construction, we applied this analysis method to Android Smart Lock and Windows Dynamic Lock, and we discovered the following security vulnerabilities:

- **Android Smart Lock:** we discovered a new MAC spoofing attack vector against Android Smart Lock, caused by an incomplete fix that counters the attack in 2015 [7, 8]. The incomplete fix allows two state transition paths for passing the device authentication; one path checks the security of connectivity, however, the other path still misses the required security checks. As a result, an attacker may unlock the smartphone without requiring the passcode.
- **Windows Dynamic Lock:** We discovered new MAC spoofing and device class spoofing attack vectors against Windows Dynamic Lock. In particular, internal authentication state transition of Windows Dynamic Lock relies on insecure properties of a Bluetooth connection, namely, the MAC address and Class of Device. As a result, an attacker may keep the computer unlocked while the user is unattended.
- We also discovered ***a common problem*** in proximity authentication via Bluetooth; that is, RSSI [9] is not a trusted property of a Bluetooth connection. Thereby, an authentication method that is solely relying on the RSSI value for proximity checking is insecure, and we illustrate a potential attack to this insecurity.

Additionally, we summarize our contributions in this paper as follows:

- We devise a new method for analyzing the security of Bluetooth-based device proximity authentication, based on modeling and analyzing the state diagrams of authentication implementations.
- We discover three new vulnerabilities in Bluetooth-based device and proximity authentication implementations, in the latest version of both Android and Microsoft Windows 10.
- We build and release a tool to check whether the authentication mechanism is secure or not against known and newly discovered attacks.
- We present potential mitigations to the attack and recommend best practices to follow when implementing the Bluetooth-based device proximity authentication mechanism.

2 Device/Proximity Authentication via Bluetooth

This section provides background in device and proximity authentication primitives in Bluetooth and introduces two implementations, namely, Android Smart Lock and Windows Dynamic Lock.

2.1 Bluetooth as a Secure Communication Channel

Bluetooth [3–6] is a wireless standard for exchanging data between computers, mobile, and other small devices. To protect data exchange, the standard provides several methods including encryption and pairing. On one hand, encryption in Bluetooth guarantees the confidentiality and integrity of trasmitted data. On the other hand, pairing in Bluetooth provides authentication of two endpoints. By mixing those two methods, two endpoints can establish a secure communication channel by authenticating each other as well as sharing a secret for enabling encryption.

2.2 Bluetooth as an Authentication Primitives

Identifying Device via Bluetooth. The pairing and encryption in Bluetooth could be extended as a method for providing device authentication. A naive implementation could authenticate a device by checking device specific IDs, such as the MAC address. Additionally, paired endpoints can further authenticate shared secrets by checking if devices has established a secure communication channel or not [6].

Measuring Device Proximity via Bluetooth. In addition to device authentication, Bluetooth could be extended to measure the proximity of two endpoint devices. In particular, an endpoint can measure the Received Signal Strength Indication (RSSI) [9], a value which will degrade as the distance of two endpoints gets further away, and is able to use this as an indirect indicator for the proximity of two devices. A popular use case of this primitive is iBeacon [10], which measures the distance of two endpoints indirectly and provides services based on the proximity.

2.3 Bluetooth Authentication Applications

By utilizing the primitives, popular software vendors such as Google and Microsoft offer the device proximity authentication method in their operating systems. In the following, we introduce Android Smart Lock and Windows Dynamic Lock, which of each is Bluetooth-based device authentication in Android and Windows, respectively.

Android Smart Lock [1]**.** Starting from version L (Android 5.0), Android adopts Bluetooth-based device authentication as its device unlock feature. The purpose of this feature is to replace default lockscreen authentication with a device authentication, for the sake of a better usability. With Android Smart Lock, a user can set a Bluetooth-enabled device, such as wireless headset, as a trusted device. The authentication is done by checking the connectivity between the device and the smartphone. In case if the connection is valid, the smartphone will not ask user for the lockscreen and unlock itself automatically. As a result, a connected, trusted Bluetooth device can act as an authentication token while the connection is intact.

Windows Dynamic Lock [2]**.** Microsoft Windows 10 (1703) adopts Bluetooth-based device proximity authentication for implementing its security feature, Windows Dynamic Lock. The purpose of Dynamic Lock is to lock the computer while the user is unattended, to prevent any unauthorized user access to the computer. In this regard, this feature complements existing authentication by offering an additional layer of security. To enable this feature, a user can register a smartphone device as a trusted device. Later, when the computer detects any of registered smartphone moves away from it, the computer regard this as an event that "user is missing", and thereby, lock the device to protect it from unauthorized access. The implementation relies on device authentication as well as proximity checking based on measuring Receive Signal Strength Indicator (RSSI).

3 A Known Weakness in Android Smart Lock

In 2015, Martin Herfurt [7] reported a MAC spoofing vulnerability in Android Smart Lock, and Beccaro and Collula [8] demonstrated the attack at the ZeroNights conference. The vulnerability stem from two facts that 1) Android Smart Lock authenticates device by checking the MAC address of the registered device, and 2) it accepts unencrypted connection via Service Discovery Protocol (SDP).

To exploit the vulnerability, an attacker may collect the MAC address of devices near the victim device. Once the MAC address of trusted device is captured, the attacker may broadcast the connection via service discovery protocol (SDP). Because SDP does not require any authentication, nor Android Smart Lock checks the authenticity of the device, the attacker can make a connection with the phone, and Android Smart Lock believes that the trusted device is with the user.

Through the responsible disclosure channel of Google, the vulnerability was reported in January 2015, and Google Vulnerability Reward Program has acknowledged the finding. Note that Google fixed this vulnerability in April 2015 (version 5.1.1, Lollipop), and our findings are different with this attack because our attack to Android Smart Lock works on the patched, latest version of Android (version 9, Android Pie).

4 Bluetooth Security 101

Before getting into the details of the security analysis of Bluetooth-based device proximity authentication methods, we give a background of Bluetooth as follows.

Bluetooth Security Features. According to the specifications [3–5] and the NIST Guide to Bluetooth Security [6], the Bluetooth standard includes the following five security features:

- **Pairing:** a process of creating and exchanging one or more shared secret keys between two endpoints.
- **Bonding:** an action of storing the shared secret keys exchanged during the pairing process, for using in subsequent connections.
- **Device Authentication:** a process of verifying the authenticity of the endpoints, by checking that the two endpoints have the same secret keys.
- **Encryption:** a function that guarantees the confidentiality of messages transmitted over the connection.
- **Message integrity:** a function that guarantees the integrity of messages against forgery attacks.

Security Mode Levels	FIPS Algorithms	MITM Protection	Encrypted
4	●	●	●
3	×	●	●
2	×	×	●
1	×	×	▲
0	×	×	×

Table 1: BR/EDR/HS Security Mode 4 Levels Summary, extracted from the NIST Guide to Bluetooth Security [6].

Security Mode Levels. On top of the five security features, the Bluetooth standard defines the Security Mode to determine how to provide these for protecting Bluetooth communication from potential attackers. Table 1 summarizes provided security guarantees per each Security Mode levels (0–4). According to the guideline and the table, NIST recommends the community to use the Security Mode 4 for secure communication; otherwise, the communication might not be secure. These Security Mode Levels are defined by the service-level-enforced security mode, whose security mechanisms are applied differently depending on the service that is on-going with a connection. For instance, the Service Discovery Protocol (SDP) works in the Security Mode Level 0, and the level does not guarantee any of authenticity, integrity, and confidentiality of the connection. In contrast, services other than the SDP, such as the Telephony Control Specification (TCS) the Radio Frequency Communication (RFCOMM), Object Exchange (OBEX), etc., can be configured as the Security Mode Level 4. The Security Mode Level 4 can be achieved only if the connection supports the five security features (listed above), and only in this level, each endpoint can trust the authenticity of the other endpoint and message confidentiality and integrity over the connection.

Bluetooth Connection Lifecycle. Figure 1 illustrates the lifecycle of a Bluetooth connection as a state diagram. A connection starts with a *Standby* state, which indicates that the connection has been not made yet. To connect with a device, the connection status gets into either the Inquiry state or the Page state. The *Inquiry* state is for searching for available devices for a connection, and this step is required if a connecting device does not know which device to connect, and it can be skipped otherwise. Once the device to connect is determined, the connection state moves to the *Page* state, which one device makes a connection to the device. Next, connection state moves to the *Link Establishment* state, which devices making a connection via the Link Manager Protocol (LMP). Note that there is no authentication up to this state transition point. After this, the link connection can be protected by the security features of Bluetooth

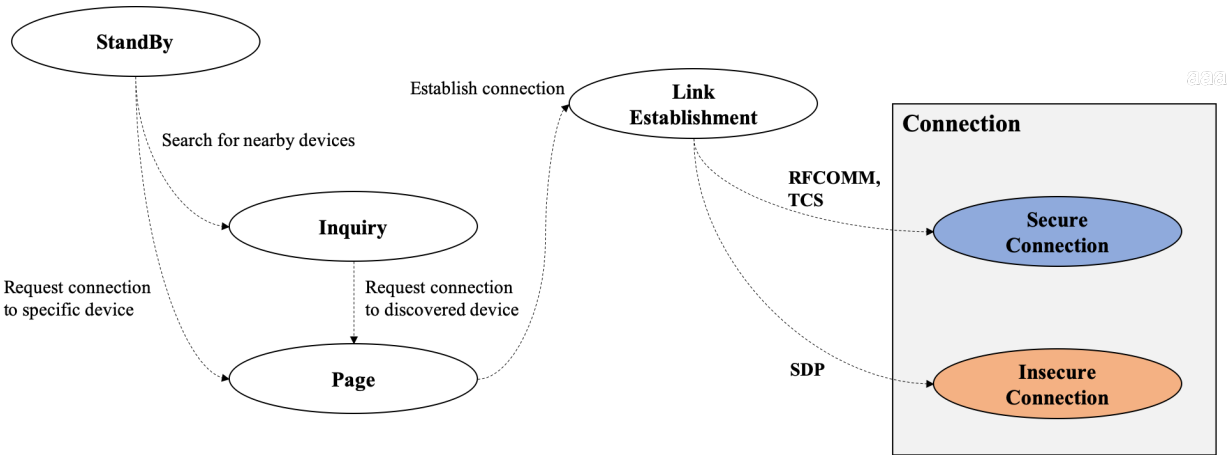

Figure 1: A state diagram of the lifecycle of a Bluetooth connection.

depending on the protocol the connection talks. For instance, use of the Service Discovery Protocol (SDP) results in making an *Insecure Connection* does not utilize any of security features of Bluetooth. In contrast, use of other protocols such as RFCOMM or Telephony Control Specifications (TCS), which relies on those five security features of Bluetooth, results in making a *Secure Connection* and having authenticity, confidentiality, and integrity guarantees over the connection.

Other Properties Used in Device Proximity Authentication Methods. In addition to making secure connections, device proximity authentication methods may utilize other properties from the connection. In the following, we introduce two properties, Class of Device (CoD) and Received Signal Strength Indicator (RSSI).

Class of Device (CoD). Class of Device (CoD) in Bluetooth is a number that describes the type of device. CoD is comprised of the major and minor device class, and the major device class is one of the followings: Miscellaneous, Computer, Phone, Lan access, Audio, Peripheral, Imaging, Wearable, Toy, and Unclassified. The minor class number can further specify device type, for example, `COD_COMPUTER_MINOR_LAPTOP` describes the device as a laptop computer, and `COD_PHONE_MINOR_SMART` describes the device as a smartphone. This information is transmitted in the the device discovery phase in the Inquery State (an unprotected connection). An example use of CoD in authentication is that Windows Dynamic Lock utilizes this information to check if the paired device is a smartphone.

Received Signal Strength Indicator (RSSI). RSSI is a measurement of the power present in a received signal via radio telecommunication. Measured as relative values, RSSI can tell the strength of the received signal over Bluetooth from the other endpoint. The link between the RSSI and proximity authentication is that the changes in RSSI can be used for detecting the distance between two endpoints of a Bluetooth connection [9]. Unfortunately, the value comes from the OSI Layer below level 2, so it can be measured regardless of the connection status, and it is not protected with authenticity, confidentiality, and integrity guarantee.

Properties	Secure?	Authentcity	Integrity	Confidentiality
MAC Address	Insecure	×	×	×
Class of Device	Insecure	×	×	×
RSSI	Insecure	×	×	×
An SDP Connection	Insecure	×	×	×
A Connection in Security Mode L4	Secure	●	●	●
A Message via Security Mode L4	Secure	●	●	●

Table 2: List of properties of a Bluetooth connection and its provided security guarantees.

Summary. Table 2 summarizes trusted/untrusted properties in a Bluetooth connection. Starting from the top, the MAC address and RSSI are captured at the OSI Layer 2 or below, which is beneath the layer that makes Security Mode Level 4 connection (Layer 4, Transportation), so it is not protected by authenticity, confidentiality, and integrity guarantees.

The SDP connection and one of its parameter, Class of Device, are works in the Security Mode Level 0, and thereby, such a connection and a value are not trusted. In contrast, a connection made in the Security Mode Level 4 and a message transmitted via the connection is fully protected by the Bluetooth connection, guaranteeing device authenticity and message confidentiality and integrity.

5 Analyzing (in)security of Device Proximity Authentication over Bluetooth

5.1 Problems in Implementing Secure Device Proximity Authentication over Blueooth

To implement a secure Bluetooth-based device proximity authentication method, one need to provide clear answers for the following two questions: *Q1) How can we authenciate a device over Bluetooth?* and *Q2) How can we check the proximity of a device securely?* In the next, we present the current problems in building secure device proximity authentication over Bluetooth by answering those two questions.

Q1: How can we authenticate a device over Bluetooth? A short answer for this is that we must rely on security features in Bluetooth to securely authenticate a device. A naive implementation could utilize the MAC address, an information that is naturally available and usually unique when assuming a benign environment, to authenticate a device. Unfortunately, according to the security of authentication primitives of Bluetooth described in Table 2, such an information can easily be collected and forged by an attacker with the capability of making a Bluetooth connection to the target device. Therefore, a secure solution must not rely on insecure, untrusted properties such as the MAC Address.

Rather, one may authenticate a device via a Security Mode Level 4 connection (in Table 1 and Table 2). Delegating the device authentication to the security features provided by a Bluetooth connection make one's life easy, because the connection in the Security Mode Level 4 guarantees the authenticity of device (via pairing and exchanging shared secrets), confidentiality (via encryption) and integrity (via Mesasge Authentication Code, MAC) of the message exchange. With this construction, an attacker must bypass the security guarantees of a Bluetooth connection to bypass the device authentication.

> **Problem 1:** Device authentication methods over Bluetooth that are relying on untrusted properties of a connection, such as the MAC Address and Class of Device, are insecure.

Q2: How can we check the proximity of a device securely? A short answer to this question is that we need to check both the authenticity of the device and the proximity of the device at the same time to guarantee the security of the proximity checking. This is because the proximity checking is not a part of security features provided by Bluetooth, and solely relying on the proximity checking feature, i.e., RSSI measurement, will result in an insecure authentication of device proximity. In the next, we present few failure cases of authenticating device proximity without checking both properties to support this argument.

Existing device proximity checking methods including iBeacon and Windows Dynamic Lock are relying on mesurement values of RSSI. However, according to Table 2, RSSI is never a secure measurement in Bluetooth, thus the value could be forged by attackers.

An alternative is to check the connectivity of endpoints (e.g., endpoints are connected or not). This can be done in a secure manner if we apply the connectivity check over a Security Mode Level 4 connection. However, this method also suffers a problem that it can only check device proximity as a digital value (i.e., the device is present or not). This is because a bluetooth connection can sustain displacement of two endpoints more than 50 meters away. In such a case, one could get the information about whether the trusted device is present or not, however, cannot get the information about whether the device is near itself or not.

> **Problem 2:** Device proximity authentication methods over Bluetooth must check both device authentication and device proximity at the same time, via a secure channel. Methods missing or failing in device authentication, such as solely relying on RSSI, will fail to authenticate the device securely, and thereby, the value could be forged by an attacker. Methods missing proximity checking, such as checking only the connectivity, cannot measure the distance between the endpoints, and thereby, can only provide the proximity as a boolean value (connected or not).

Our hypothesis: failing to address either Problem 1 or 2 would result in an insecure authentication Based on our observation of two problems in device proximity authentication over Bluetooth, we propose a hypothesis that any failure in addressing either **Problem 1** or **Problem 2** would result in an insecure authentication. In the next, we propose

our method to prove our hypothesis by modeling authentication state diagrams and analyzing the security of them.

5.2 Proposed Analysis Approach

To check if an authentication implementation addresses two main questions correctly or not, we require analysis on two logics of the implementation. One analysis is for checking if the device authentication is secure, and the other analysis is for checking if proximity authentication is bound with a secure device authentication. In the following, we illustrate steps of our analysis approach. We suggest to analyze the Bluetooth-based proximity authentication following:

1. Define trusted/untrusted properties in a Bluetooth connection, as in Table 2. The primary rule in using such properties is that the authentication decision must rely on trusted properties.
2. Model two state diagrams of the authentication implementation; one is for managing a Bluetooth connection and the other is for the logic that processes the device and proximity authentication and access authorization. This can be done by reviewing both developer's and user's manual or applying reverse-engineering to the implementation.
3. Analyze two state diagrams for potential vulnerability. Specifically, we regard any state transition to a successful authentication state initiated by any untrusted property in a Bluetooth connection as a potential vulnerability.

5.3 A Working Example: Analysis of the Attack to Android Smart Lock in 2015

We apply our proposed analysis method to the known attack (in §3) against Android Smart Lock. For the first step of defining trusted/untrusted properties of a Bluetooth connection, we reuse our findings in Table 2. For the second step, we applied reverse engineering to the Google Mobile Service APK file to extract which type of Bluetooth connection that Android Smart Lock utilize, which properties are being used in authentication, how authentication decisions are made, etc., to build state diagrams for the authentication logic as well as Bluetooth connection logic. In the next, we give an example of applying our analysis as the third step to re-discovering the known attack.

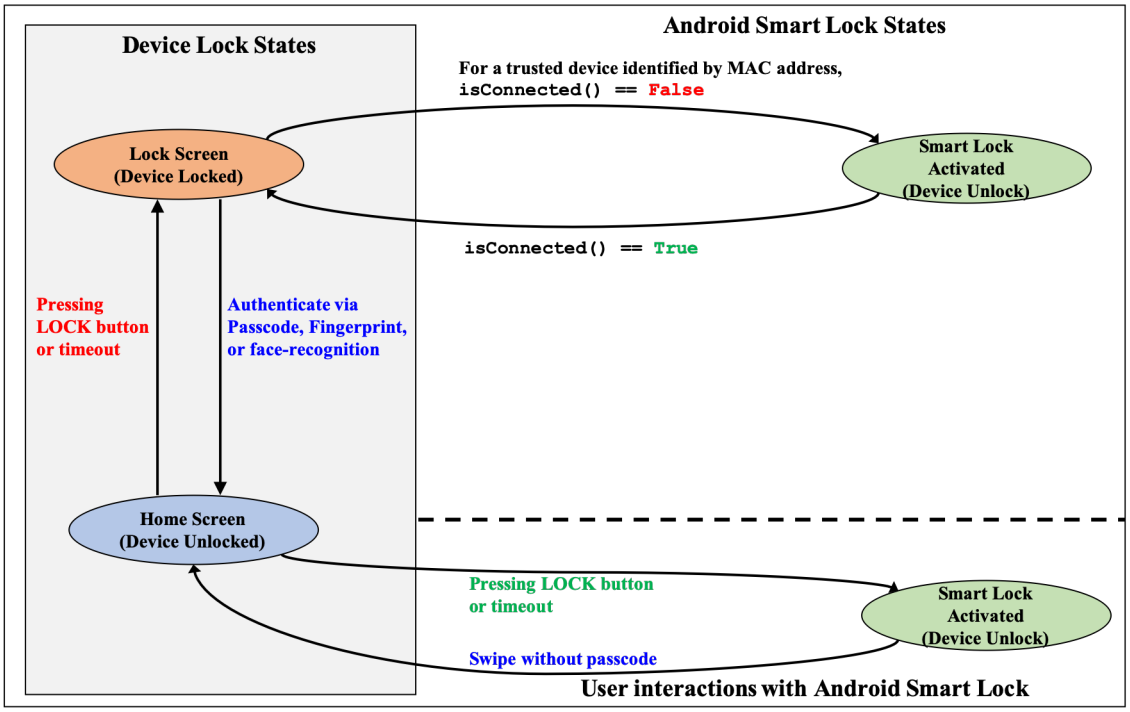

Figure 2: The state diagram of the logic that authenticates a device and authorizes access in Android Smart Lock, before applying a patch that counters the attack in 2015. On identifying the connection with trusted device, the logic in Android Smart Lock only check if a connection from registered trusted device (identified by MAC address) is established (or not), and does not check if it is a secure connection or not.

Authentication Logic of Android Smart Lock. Figure 2 shows how Android Smart Lock authenticates a device and authorizes an access to the smartphone. The left side (grey box) indicates a normal lock screen operation. The device is normally at the *Lock Screen (Device Lock)* state, and only if an explicit authentication is presented by the user (via the pattern lock or the passcode), then the device can be in the unlocked and the state changes to the *Home Screen* state. On top of this, enabling Android Smart Lock adds a new state, *Lock Screen (Device Unlock)*, on the right side. When the smartphone holds a live connection with a trusted device, the state of the authentication changes to the *Lock Screen (Device Unlock)*. In this state, the device will not ask user for the passcode, so swiping up on the screen changes the device state to *Home Screen*. In case if the connection is terminited, the device state reverts back to the state without Android Smart Lock (either Lock Screen with Device Lock or Home Screen).

Authentication Logic Summary: Android Smart Lock will be in the *Lock Screen (Device Unlock)* state if it detects a connection from a trusted device. In the state, the smartphone will not ask the passcode for accessing the home screen.

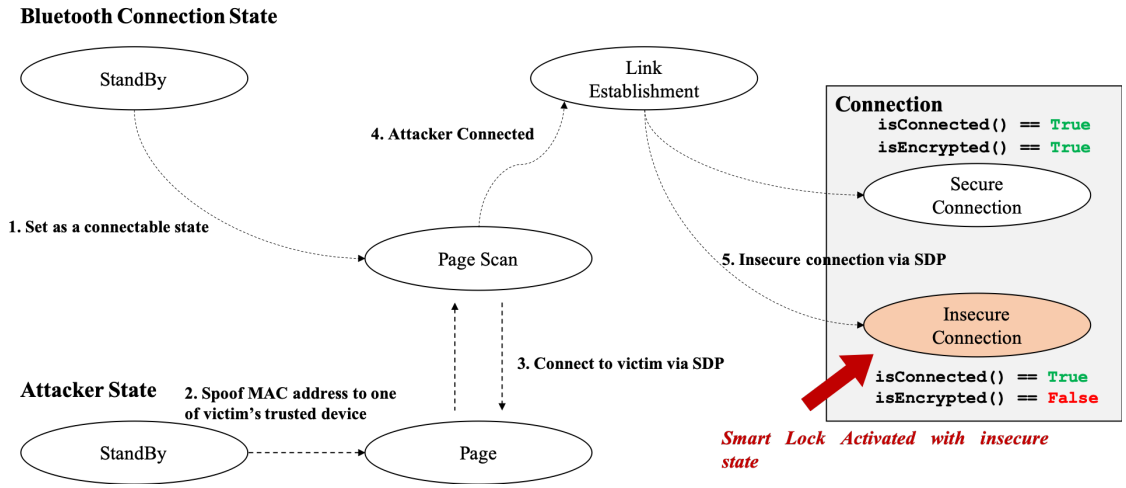

Figure 3: The state diagram of the logic that manages a Bluetooth connection in Android Smart Lock. Because Android Smart Lock checks only for the flag `isConnected() == True`, the attacker may activate Android Smart Lock by establishing an insecure SDP connection.

Connection Management Logic of Android Smart Lock. Figure 3 illustrates how Android Smart Lock utilizes a Bluetooth connection for its device authentication. In analyzing this diagram, we assume that the user has already paired and registered a trusted device, and our analysis starts with the *Standby* state that connection with the device has not started at all. From the *Standby* state, Android Smart Lock will *page* the trusted device because the device has already been registered to it. Note that this paging is relying on the MAC address for discovering registered device. Next, if the trusted device shown up, the Bluetooth connection state will be changed to the *Link Establishment* state automatically. After that, if the link establishment is successful, the device and Android Smart Lock will establish a connection. In establishing the connection, if the device is the registered, trusted device, then a *Secure Connection* will be made (and also have the transiton into the state). Otherwise (not registered nor trusted device), an *Insecure Connection* will be made, if the device has not paired with Android Smart Lock yet.

Connection Management Logic Summary: Android Smart Lock will page the device, if found, it tries to make a connection with it. Depending on the authenticity of the device, the authentic trusted device will make a *Secure Connection* with Android Smart Lock. In contrast, a fake (un)trusted device will make an *Insecure Connection* with Android Smart Lock.

The Vulnerability. We can re-discover the vulnerability by analyzing the two state diagrams. From the analysis of the connection management logic, we have identified that Android Smart Lock makes a connection to registered, trusted device via paging. In case if the device is authentic trusted device, a secure connection will be made. Otherwise, if the device is a fake one, an insecure connection will be made.

Unfortunately the authentication management logic is affected by **Problem 1** . What Android Smart Lock checks

is the presence of a connection with a trusted device not the security state of the connection. In other words, Android Smart Lock will be activated even if the connection is insecure one because the authentication logic never checks the Bluetooth connection status.

Re-discovered attack: Although any device can forge the MAC address and make an insecure connection with Android Smart Lock, the authentication logic activates Android Smart Lock without checking the Bluetooth connection status. Therefore, an insecure connection can activate Android Smart Lock, and an attacker may successfully unlock the phone without the passcode.

6 New Vulnerabilities in Android Smart Lock and Windows Dynamic Lock

In this section, we introduce *three* new vulnerabilities that we found by applying our analysis method to both Android Smart Lock and Windows Dynamic Lock.

6.1 Android Smart Lock – bypassing `isEncrypted()` check

The known attack and the patch. The known attack by Martin Herfurt exploits a MAC spoofing attack to bypass device authentication. In response to the attack, Google released the patch that checks if the device is connected with the phone via an encrypted connection, by checking the return value of `isEncrypted()`; Android Smart Lock will be activeated if the function call returns `TRUE`, and this is the Google's expectation in the latest version of Android.

Summary of newly discovered attack. An attacker may bypass the check guarded by `isEncrypted()` when the attacker launches a MAC spoofing attack while the phone is in the unlocked state (*i.e.*, user is using the phone). The root cause of the vulnerability is that the patch made in 2015 failed to guard all state transitions to the successful authentication. Specifically, the connection management logic of the patch contains *two* possible state transitions to the successful authentication; one is guarded by `isEncrypted()`, and the other is not blocked, and thereby, attackers may still launch the MAC spoofing attack. Figure 4 shows patched checks for both `isConnected == True` and `isEncrypted() == True` when the device is in locked state. However, the logic opens an alternative path (marked in blue arrows) that checks only for `isConnected() == True`, and thereby, failed to apply security checks for all available state transition paths.

Authentication Logic Analysis. Figure 4 shows the state diagram of the logic that authenticates a device and authorizes access in patched version of Andorid Smart Lock. The difference between Figure 4 and Figure 2 is that the patch in 2015 added a security check to validate the capability of the device (e.g., knows shared secret). This is done by checking if the connection is encrypted or not because only devices that knows shared secret can only establish an encrypted connection. In particular, the patch checks for `isEncrypted() == True` to not only identify the trusted device by the registered MAC address but also check if the device and the phone has made an *encrypted* connection. We believe that the purpose of this patch is to utilize trusted information in a Bluetooth connection for authentication, which is a good secure practice.

Authentication Logic Summary: The patched version (for the 2015 attack) of Android Smart Lock will check not only the establishment of a connection from a trusted device (via MAC address) but also the establishment of an encrypted connection (via shared secret) to check if the connected device has been securely paired with the Smartphone before. However, such check is only applied to the case when the smartphone is in *Lock Screen* and is not applied to the case when the smartphone is in unlocked state, i.e., *Home Screen*. In the latter case, Android Smart Lock is activated by checking only the connection establishment (no checking for the encrypted connection), and thereby, an attacker may launch the MAC spoofing attack to bypass device authentication.

The Vulnerability. From the analysis of the connection management logic, we have identified that the patch failed to guard all possible connection state transition from *Page Scan* to *Link Establishment* (see Figure 3), thus it is affected by **Problem 1** . Specifically, the patch allows an alternative path to a successful authentication without passing the check done by `isEncrypted()`. That is, the patch does not apply the check if the phone has been already unlocked (*i.e.*, , while the phone is being used by the user). In this case, an attacker can successfully made a connection from a fake device (with a forged MAC address) to the phone without having `isEncrypted()` check. Because the authentication check is

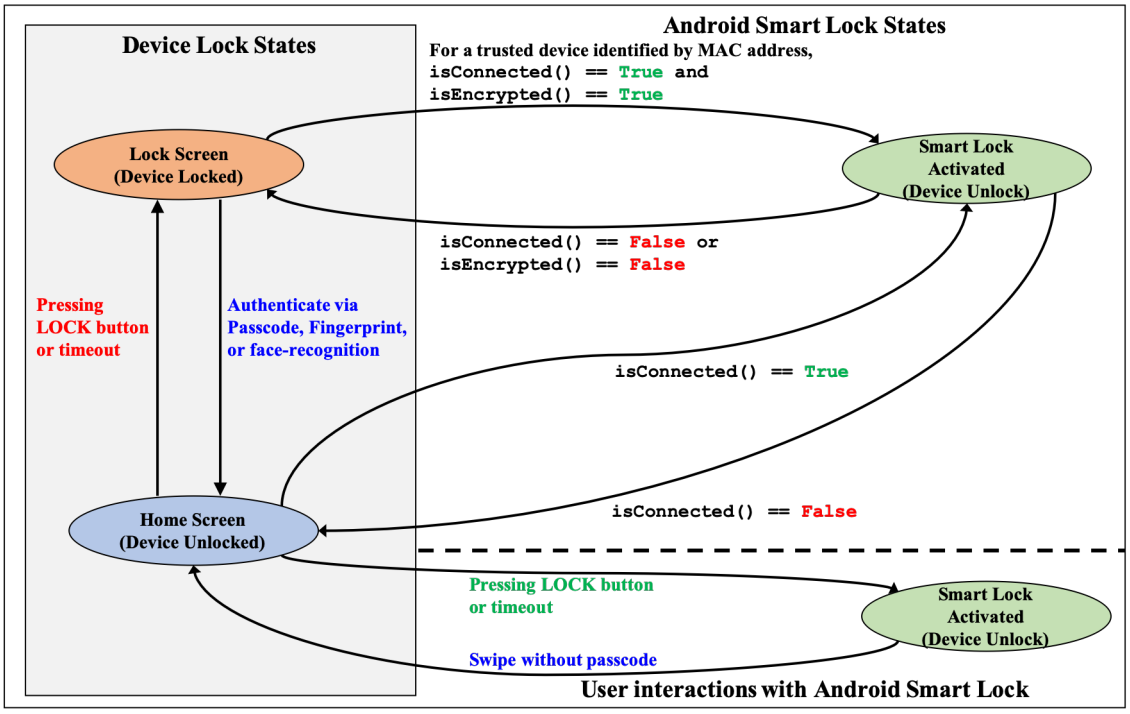

Figure 4: The state diagram of the logic that authenticates a device and authorizes access in Android Smart Lock, after applying a patch that counters the attack in 2015. In addition to checking `isConnected() == True` in Figure 2, the patched version also checks for `isEncrypted() == True` to verify the connection is from a paired, trusted device. However, checking of encrypted connection happens only when the Smartphone is in Locked state (for main screen). Unlike in locked state, when the Smartphone is unlocked (e.g., user is using the phone), Android Smart Lock only checks for `isConnected() == True`, by missing checks for `isEncrypted() == True`.

done at the Stage 4 and not afterwards, the attacker can successfully get to the *Connection* state, which Android Smart Lock regards that a trusted device has established a connection with the phone.

A new vulnerability discovered from Android Smart Lock: The patch misses checking `isEncrypted() == True` when the Smartphone is unlocked (e.g., being used by the user). Because Android Smart Lock allows activation of it via making an insecure connection to the Smartphone (to pass `isConnected() == True`), attackers can bypass device authentication and can successfully unlock the Smartphone without passcode.

Attack Scenario. Based on our analysis, we present an attack scenario of using Android Auto (a car as a trusted device) that an attacker may re-enable Android Smart Lock by launching the MAC spoofing attack (faking the MAC as that of the Bluetooth device of the car). We assume that an attacker with Bluetooth capability is near the user at the parking lot, and the user is leaving the car while using her phone.

Assumption: The user has set her car as a trusted device to use Android Auto with the phone (victim device). An attacker is waiting for the user at the parking lot.

Step 1: The attacker acquires the MAC address of the Bluetooth device of the car. The attacker can achieve this by capturing Bluetooth packets over the air.

Step 2: The user parks the car, turns off the car, and gets out from the car. The Bluetooth connection between the car and the victim device (phone) is disconnected.

Step 3: The attacker tries to make a connection with the victim device via Service Discovery Protocol (insecure connection).
The connection will not activate Android Smart Lock if the victim device is locked because it may not pass the check done by `isEncrypted()`.

Step 4: Once the user unlocks the victim device for some reason (*e.g.*,, checking SMS message, e-mail, etc.), then the insecure connection between the attacker and the victim device will activate Android Smart Lock (no check for `isEncrypted()`).

Result: the victim device will never be locked while the attacker's connection is active.

6.2 Windows Dynamic Lock - Bypassing trusted device authentication

Identifying states Windows Dynamic Lock works as shown in Figure 5. When user first logins by either of typing password, input fingerprint, or passing face-recognition authentication, Windows search for the near by Smartphone that has already been paired with the PC (see the bottom arrow of Figure 5). In doing this, Windows will only send this request to the list of previously paired device. With this request, the near by paired Smartphones will serve as trusted devices. To facter out non-smartphone devices, Windows checks the class-of-device (CoD) field of the connection message. Only the devices that asserts itself as Smartphone at the connection (when a user's login happens) will serve as trusted devices. After this, Windows Dynamic Lock will keep track of connection status of those trusted devices, and if it is disconnected or its RSSI gets lower than -10db, then Windows Dynamic Lock will regard that as user is moving away from the device, and there by, lock the device.

In summary, Bluetooth Connection is used for identifying and authenticating trusted Smartphones (by checking previously paired MAC addresss and class of device), and RSSI or connection status (i.e., disconnected or not) is used for measuring the proximity of the phone with the PC.

Authentication Logic Analysis. Figure 5 shows the available state transitions in the authentication logic of Windows Dynamic Lock. When user logins to the PC (the transiation at the bottom), the PC will ping previously paired device (with PC) to make them a Bluetooth connection to the PC. The identification of the device is done by matching the MAC address. Additionally, Windows Dynamic Lock will accept and track the connection only from Smartphone devices. The identification of the Smartphone is done by checking the class-of-device (CoD) field of the connection request comes from each device. When user moves away from the PC (the transition at the top), the connection with the trusted device can either be disconnected or be with a weak connection status (e.g., RSSI $<$ -10db). If detected, Windows Dynamic Lock will regard that event as user is unattended, and thereby, lock the device immediately.

Authentication Logic Summary: Windows Dynamic Lock will make connections with previously paired Smartphones when user logins to the PC. To identify a paired device, it utilizes the MAC address field from the Bluetooth connection. To identify a Smartphone, it utilizes the class-of-device (CoD) field from the Bluetooth Connection. However, because both the MAC address and class-of-device are insecure data of the connection (i.e., attackers can spoof those easily), the logic may suffer from **Problem 1** .

Connection Management Logic Analysis.

Figure 6 shows the state diagram of the logic that manages a Bluetooth connection in Windows Dynamic Lock. By combining state transition conditions in the authorization logic (Figure 5) and the connection management logic (Figure 6), we can test our hypothesis to discover new attacks.

First, Windows Dynamic Lock is susceptible to **Problem 1** . Because Windows Dynamic Lock is activated at the *Link Establishment* state, which is a insecure state of a Bluetooth connection, any attacker that can spoof the MAC

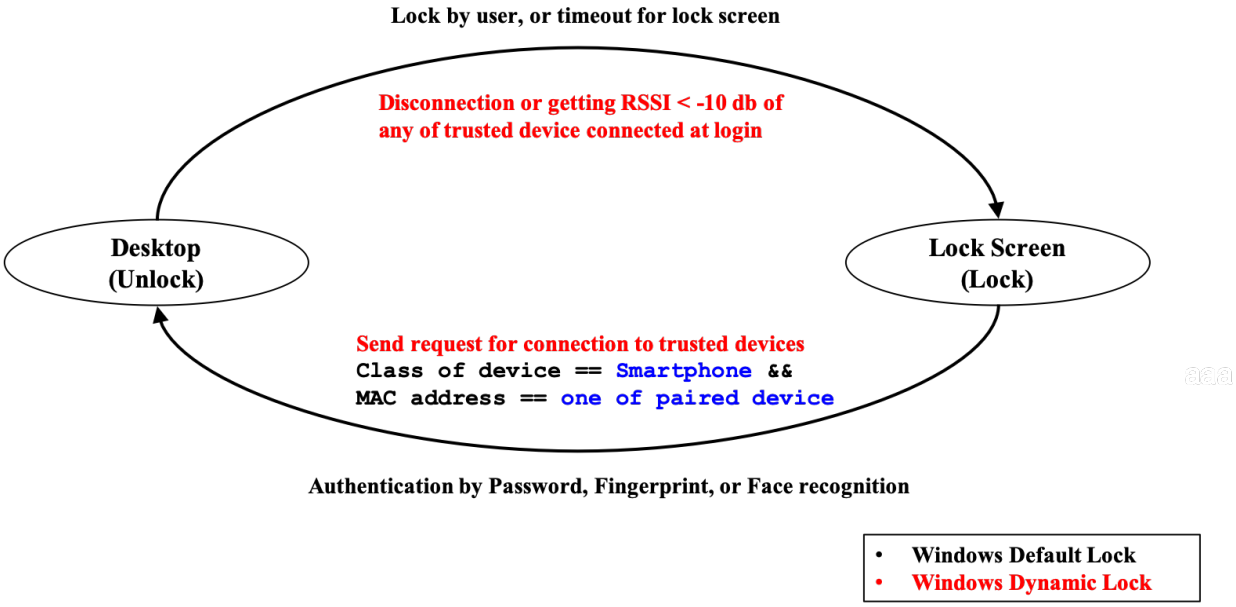

Figure 5: The state diagram of the logic that authenticates a device and tracking proximity of the device in Windows Dynamic Lock. When user logins to the PC (the transiation at the bottom), the PC will ping previously paired device (with PC) to make them a Bluetooth connection to the PC. Additionally, Windows Dynamic Lock will accept and track the connection only from Smartphone devices. When user moves away from the PC (the transition at the top), the connection with the trusted device can either be disconnected or be with a weak connection status (e.g., RSSI $<$ -10db). If detected, Windows Dynamic Lock will regard that event as user is unattended, and thereby, lock the device immediately.

address of one of trusted device can establish such a connection (thus activates Windows Dynamic Lock). Although it checks for the class-of-device field to verify if the device that makes a connection is a Smartphone, the field is also insecure, in other words, can easily be spoofed by the attacker.

Second, Windows Dynamic Lock is susceptible to **Problem 2** . This is because Windows Dynamic Lock measures the signal strength (RSSI) regardless of the state of the connection, in other words, measuring the proximity even for the insecure connection. RSSI can easily spoofed by lowering the power of the transmission (RSSI decreases) or repeating messages with higher power (RSSI increases), and because Windows Dynamic Lock does not check the connection status of the device, the Lock is susceptible to attacks.

Based on our analysis, we developed the following two attacks to Windows Dynamic Lock:

Vulnerability 1: denying signal decay. We can launch the MAC address and class-of-device spoofing attack to Windows Dynamic Lock to make them to fail to detect signal decay (*i.e.*, detecting user is moving away from the PC). To this end, the attacker may capture one of the MAC address of a victim's Smartphone over the air. After that, the attacker can set its MAC address as captured one and can set its class-of-device as Smartphone. And then, the attacker keep tries to connect to the PC via insecure SDP connection.

Under this condition, the connection request will be failing because the PC has already been connected with the victim's Smartphone. However, when the user moves away from the PC, the signal strength will be weaken, and at certain point, the original connection will be disconnected. In such a case, the PC will retry the connection (as a default connection fault-tolerant behavior), and at this time, it will be re-connected to the attacker's device. This is because the attacker is sending the same information (MAC address and class-of-device) that of victim's Smartphone. Finally, because Windows Dynamic Lock does not check the security of the connection, the connection check will be bypassed, and thereby, Windows Dynamic Lock cannot detect that the user is moving away from the device.

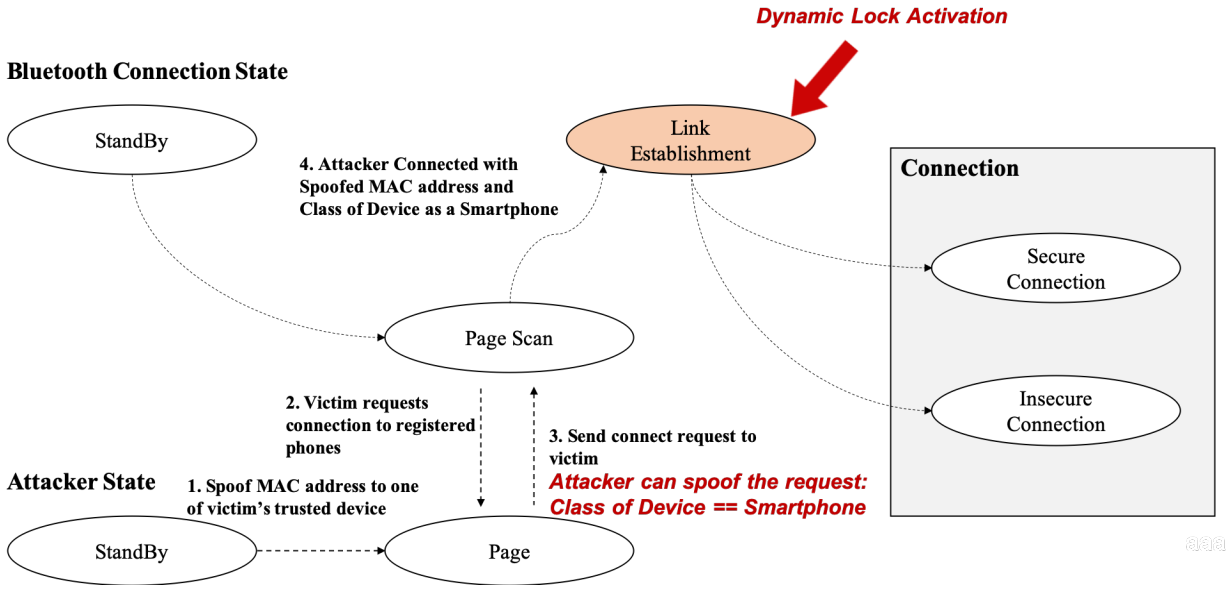

Figure 6: The state diagram of the logic that manages a Bluetooth connection in Windows Dynamic Lock. Windows Dynamic Lock is activated before device establishes any insecure or secure connection to the PC, and thereby, susceptible to our new attack.

Vulnerability 1 Summary: Launching the MAC address/CoD spoofing attack may establish the connection to the PC when the signal strength between the user's Smartphone and the PC is weak. In such a case, Windows Dynamic Lock will regard the Smartphone is near the PC (by checking the new connection), and thereby, fail to lock the device even if user is unattended at the PC.

Attack Scenario:

Assumption: The user is in the office space, and the attacker is running a malware in one of the laptop near the user's PC.

Step 1: Attacker obtains the MAC address of victim's Smartphone. This could be done by measuring Bluetooth traffic over the air, or measuring the MAC address of Smartphone's Wi-Fi and apply +1 to the address (works with many smartphones).

Step 2: Attacker will keep sending SDP connection requests to user's PC by spoofing the MAC address as that of victim's Smartphone and its class-of-device as a Smartphone. If user's smartphone is closer to the PC than the attacker's laptop, then the connection will never be made.

Step 3: When user moves away from the PC, the connection between the smartphone and the PC will eventually be weaken. In such a case, the attacker's connection request will be accepted by the PC.

Step 4: Now the PC regards that the victim's trusted device is near the PC, thereby, the PC will not be locked even if the user is away.

Vulnerability 2: denying trusted device connection. The attacker may launch another type of MAC address/CoD spoofing attack against Windows Dynamic Lock. Unlike the vulnerability 1, this cuts off the connection of trusted devices when user logs in to the PC, by spoofing the class-of-device of Bluetooth device as non-Smartphone device. To this end, an attacker may capture the MAC address of victim's Smartphones. After obtaining this, the attacker will keep sending SDP connection request with the MAC address of victim's Smartphone but with the class-of-device as non-Smartphone, e.g., a Bluetooth headset.

In such a case, when the user tries to login to the PC, Windows Dynamic Lock will regard the connection from the MAC address of the Smartphone as a Bluetooth headset, so the device is not being tracked by Windows Dynamic Lock. As a result, even if the user moves away from the PC, Windows Dynamic Lock fails to lock the device.

Vulnerability 2 Summary: Launching the MAC address/CoD spoofing attack may establish the connection to the PC before user's Smartphone establishes a connection. In such a case, attacker can set its class-of-device as non-Smartphone (i.e., a Bluetooth headset), then Windows Dynamic Lock will not track the connection status of the device, so Windows Dynamic Lock will not be activated (thus not being locked even if the user moves away).
Attack Scenario:

Assumption: The user is in the office space, and the attacker is running a malware in one of the laptop near the user's PC.

Step 1: Attacker obtains the MAC address of victim's Smartphone. This could be done by measuring Bluetooth traffic over the air, or measuring the MAC address of Smartphone's Wi-Fi and apply +1 to the address (works with many smartphones).

Step 2: Attacker will keep sending SDP connection requests to user's PC by spoofing the MAC address as that of victim's Smartphone and its class-of-device as a Bluetooth headset (not a Smartphone). This will make the PC regard the connection is from non-Smartphone device.

Step 3: Windows Dynamic Lock fails to find previously paired Smartphone from Bluetooth connections, so it will not be activated.

Step 4: When user moves away from PC, the PC will not be locked because Windows Dynamic Lock is not activated.

6.3 Windows Dynamic Lock - Proximity Measurement

In addition to the Device Authentication, Windows Dynamic Lock is susceptible to the attack for its measuring logic of the Proximity. That is, Windows Dynamic Lock utilizes RSSI as an indicator of the proximity of the device. In order to obtain RSSI of the peer device, the PC will obtain this from the link manager located in the controller using HCI request. But because RSSI is not bound with secure messages, and the controller simply uses the MAC address to identify the peer's RSSI (see Figure 5 and Figure 6), Windows Dynamic Lock can't determine whether it is really from the trusted peer device. In order words, Bluetooth connection can guarantee the integrity of the received message and the peer, but nothing about RSSI, thus using RSSI value measured by an insecure connection is not secure.

7 Potential Mitigations to the Attacks

To mitigate new attacks that we have discovered against Bluetooth-based proximity authentication, we propose several countermeasures.

Applying analysis in the software development lifecycle (SDL). We highly recommend incorporating our state diagram based analysis method to the software development lifecycle of the authentication method. By analyzing state diagrams of the authentication logic and the connection management logic, one may check if any state transition is made by any untrusted properties of a Bluetooth connection. If so, please replace the transition trigger to a trusted property.

Completely cut-off insecure paths. Once the analysis is done, please make sure to identify and fix all insecure paths of state transitions. A bad example of not applying this kind of analysis nor completely cut-off insecure paths is the patch done by Google in 2015. In the fix, Google knew that the root cause of the vulnerability is that Android Smart Lock relies on an insecure Bluetooth connection. In response to the attack, Google fixed that by checking if the connection is secure via invoking `isEncrypted()`. However, the fix failed to cut-off the entire insecure paths in the state transition, resulting in a new vulnerability. Thereby, apply a thorough analysis to identify and fix all insecure paths of state transitions.

Binding insecure properties with a secure connection. Using only the trusted properties of a connection would resolve most of the issues, but not all. In such a case, we may take advantage of a secure connection to make the use of insecure property secure. For instance, RSSI, a relative value that could tells the displacement of endpoints, is never a secure property. Because the root cause of the vulnerability stems from that the authentication logic utilizes untrusted properties of a Bluetooth connection, we may solely utilize the value. Fortunately, because the RSSI value is physically bound with a connection, measuring the RSSI value from a secure connection will prevent the device forgery attack.

Bluetooth stack has already practiced this principle for making other untrusted properties trusted; that is, we can trust the MAC address if the connection is secure.

7.1 Vulnerability Detection Tool

We also provide a tool to assess the security of Bluetooth-based device proximity authentication against proposed attacks. The tool implements two types of attacks. One is against device authentication by exporting an interface for altering untrusted properties of a Bluetooth connection (e.g., MAC address, class of device, class of service, etc.). The other is against link establishment by making an insecure connection to the victim device (via SDP).

Because the required status of the victim device for applying the attack can vary, the assessment would require additional steps to make the victim device into a specific state. In such a case, one may apply our analysis method to extract state transition diagrams and then apply the attack tool to check if the authentication implementation is vulnerable or not.

8 Responsible Disclosure of Vulnerabilites Discovered

Before the submission, we have reported all vulnerabilities that we found throughout this project via the responsible disclosure channel of software vendors.

In particular, we have reported Google (in April 2019) regarding vulnerabilities in Android Smart Lock and their mistake in patching MAC spoofing attacks. As a result, Google accepted our vulnerability report, as *Issue 130010821: Auth Bypass in Google Play Service (SmartLock) for Android*, and rewarded a bug bounty, and fixed the vulnerability.

Additionally, we have reported Microsoft (in April 2019) regarding vulnerabilities in Windows Dynamic Lock against the MAC spoofing as well as the device type spoofing attack. In response to our report, Microsoft replied to us: *If the attacker has spoofed the MAC address of the user's phone, and is continuously maintaining connection with the computer, the Dynamic Lock service will never call WinLogon to lock the device. However, there are other inactivity timers in WinLogon which are independent of Dynamic Lock. If the device has any sort of "lock/sleep after X minutes" setting, then after X minutes of inactivity, the machine will lock regardless of the state of Dynamic Lock. So there is no regression to the original security promise.*

However, we believe our attack scenario is still valid under the condition that Microsoft asserts; by exploiting accessibility attack [11], in a public office setting, an attacker may play audio saying "Cortana, open Microsoft Edge"; this audio will command the victim device to open the Microsoft Edge browser while the device is unlocked (if the voice activation of Cortana is enabled, which Microsoft recommends for Windows 10). Launching such an attack will bypass inactivity check, and thereby, the victim device may unlocked indefinitely.

9 Conclusion

Bluetooth-based device proximity authentication is widespread, however, currently deployed implementations are not secure. The insecurity stems from lacking of security features for proximity authentication in Bluetooth, and this makes the authentication logic insecure if the logic is relying on untrusted properties of Bluetooth. We address this problem by devising an analysis method for such authentication issues and demonstrating its application to popular, deployed implementations.

We conclude with summarizing the following lessons to the readers. First, use only trusted properties of Bluetooth when implementing authentication method. This includes the use of authenticated and encrypted connection (definitely not SDP) and binding a trusted property with an untrusted property if the untrusted value is required. Second, know that Bluetooth specification does not guarantee the proximity of a device. Third, understand trusted/untrusted components of Bluetooth. Fourth, Incoporate the methodology for analyzing the security of Bluetooth authentication, security analysis via model state diagram, to the development lifecycle. Fifth, check your implementation with our tools.

References

[1] Android Help, "Set Your Android Device to Automatically Unlock." https://support.google.com/android/answer/9075927?hl=en, Aug. 2019.

[2] Microsoft Windows Support, "Lock Your Windows 10 PC Automatically When You Step Away From It." https://support.microsoft.com/en-us/help/4028111/windows-lock-your-windows-10-pc-automatically-when-you-step-away-from, Mar. 2019.

[3] Telefonaktiebolaget LM Ericsson, International Business Machines Corporation, Intel Corporation, Nokia Corporation, and Toshiba Corporation, "Specification of the Bluetooth System." http://grouper.ieee.org/groups/802/15/Bluetooth/profile_10_b.pdf, Dec. 1999.

[4] Joe Decuir, "Bluetooth 4.0: Low Energy." https://californiaconsultants.org/wp-content/uploads/2014/05/CNSV-1205-Decuir.pdf, May 2014.

[5] Bluetooth SIG, "Bluetooth Core Specification v5.0." https://www.mouser.it/pdfdocs/bluetooth-Core-v50.pdf, Dec. 2016.

[6] J. Padgette, K. Scarfone, and L. Chen, "Guide to Bluetooth Security," *NIST Special Publication*, vol. 800, no. 121, p. 25, 2012.

[7] M. Herfurt, "Tricking Android Smart Lock with Bluetooth," 2015. https://mherfurt.wordpress.com/2015/05/08/tricking-android-smart-lock-with-bluetooth/.

[8] M. Beccaro and M. Collura, "Extracting the Painful (Blue)tooth." http://2015.zeronights.org/assets/files/21-Beccaro-Collura%20.pdf, Nov. 2015.

[9] Vincent Gao, "Bluetooth Blog: Proximity and RSSI." https://www.bluetooth.com/blog/proximity-and-rssi/, Sept. 2015.

[10] Apple Developer, "iBeacon." https://developer.apple.com/ibeacon/, Aug. 2019.

[11] Y. Jang, C. Song, S. P. Chung, T. Wang, and W. Lee, "A11y Attacks: Exploiting Accessibility in Operating Systems," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 103–115, ACM, 2014.